Devoir d'informatique 5

D.Malka – MPSI 2020-2021 – Lycée Jeanne d'Albret

11.06.2021

Durée de l'épreuve : 2h00

L'usage de la calculatrice n'est pas autorisé.

L'énoncé de ce devoir comporte 6 pages.

- *Si, au cours de l'épreuve, vous repérez ce qui vous semble être une erreur d'énoncé, signalez le sur votre copie et poursuivez votre composition en expliquant les raisons des initiatives que vous êtes amené à prendre.*
- *Il ne faudra pas hésiter à formuler des commentaires. Le barème tiendra compte de ces initiatives ainsi que des qualités de rédaction de la copie.*
- *La numérotation des exercices doit être respectée. Les résultats doivent être systématiquement encadrés.*
- *Les pages doivent être numérotées de la façon suivante : n°page courante/nombre total de pages.*

Exercice 1 – Base de données universitaire

Une université possède la base de données (simplifiée) relatives à ses étudiants suivantes :

```
etudiants(ne,nom,prenom,id_formation,date_naissance,adresse,salarie)
formation(id_formation,libelle_f,type)
module(id_module,id_formation,libelle_m)
resultat(ne,id_module,note,annee)
```

`ne` : numéro étudiant, type : INT `nom` : nom de l'étudiant, type : VARCHAR `prenom` : prenom de l'étudiant, type : VARCHAR `date_naissance` : date de naissance de l'étudiant, type : DATE `adresse` : adresse de l'étudiant, type : VARCHAR `id_module` :réference du module, type : INT `note` : note de l'étudiant à l'examen relatif au module, type : INT	`annee` : annee d'obtention du module, type : INT `id_formation` : réference de la formation, type : INT `libelle_m` : libellé du module , type : VARCHAR `libelle_f` : libellé de la formation , type : VARCHAR `type` : type de formation , type : VARCHAR `salarie` : indique si l'étudiant est salarié (=1) ou non (=0), type : INT

- Pour simplifier, tous les modules d'une formation ont même coefficient.
- La référence d'une formation est unique.
- Les types de formation sont « L1 », « L2 », « L3 », « M1 », « M2 ».
- Une formation dure un semestre.
- Une formation comprend plusieurs modules.
- Pour simplifier, un module ne peut pas être partagé par plusieurs formations.
- La formation est validée si la moyenne des modules relatifs à cette formation est supérieure ou égal à 10.
- Un module est validée si la note obtenu à l'examen est supérieur ou égale à 10, même si la formation dans son ensemble n'est pas validée.
- On peut donc valider sa formation sans valider chacun des modules.
- Seuls les modules non validés sont à repasser l'année suivante.
- Les notes sont entières et comprises entre 0 et 20.

— L'adresse d'un étudiant est donnée uniquement par le nom de sa ville de résidence.
— Pour simplifier, on suppose que chaque étudiant ne suit qu'une seule formation par année.

Les requêtes seront exprimés en langage SQL.

1. Requête affichant les noms et prénoms des étudiants salariés.
2. Requête affichant les numéros étudiant, les noms et prénoms des étudiants en L1.
3. Requête affichant les noms et prénoms des étudiants inscrits en M1 et ayant suivi le module « Informatique ».
4. Requête affichant le libellé et le nombre d'étudiants dans chaque formation.
5. Requête affichant la liste (numéro d'étudiant, nom, prénom) ainsi que la moyennes générales de chacun des étudiants ayant validé son année en 2015, quelle que soit la formation.

Exercice 2 – Gestion du trafic aérien

Plan de vol

Afin d'éviter les collisions entre avions, les altitudes de vol en croisière sont normalisées. Dans la majorité des pays, les avions volent à une altitude multiple de 1000 pieds (un pied vaut $30,48\,cm$) au-dessus de la surface isobare à 1013,25 hPa. L'espace aérien est ainsi découpé en tranches horizontales appelées niveaux de vol et désignées par les lettres « FL » (flight level) suivies de l'altitude en centaines de pieds : « FL310 » désigne une altitude de croisière de 31000 pieds au-dessus de la surface isobare de référence. Eurocontrol est l'organisation européenne chargée de la navigation aérienne, elle gère plusieurs dizaines de milliers de vol par jour. Toute compagnie qui souhaite faire traverser le ciel européen à un de ses avions doit soumettre à cet organisme un plan de vol comprenant un certain nombre d'informations : trajet, heure de départ, niveau de vol souhaité, etc. Muni de ces informations, Eurocontrol peut prévoir les secteurs aériens qui vont être surchargés et prendre des mesures en conséquence pour les désengorger : retard au décollage, modification de la route à suivre, etc. Nous modélisons (de manière très simplifiée) les plans de vol gérés par Eurocontrol sous la forme d'une base de données comportant deux tables :

— la table **vol** qui répertorie les plans de vol déposés par les compagnies aériennes ; elle contient les colonnes
  - `id_vol` : numéro du vol (chaine de caractères) ;
  - `depart` : code de l'aéroport de départ (chaine de caractères) ;
  - `arrivee` : code de l'aéroport d'arrivée (chaine de caractères) ;
  - `jour` : jour du vol (de type date, affiché au format aaaa-mm-jj) ;
  - `heure` : heure de décollage souhaitée (de type time, affiché au format hh :mi) ;
  - `niveau` : niveau de vol souhaité (entier).

id_vol	depart	arrivee	jour	heure	niveau
AF1204	CDG	FCO	2016-05-02	07 :35	300
AF1205	FCO	CDG	2016-05-02	10 :25	300
AF1504	CDG	FCO	2016-05-02	10 :05	310
AF1505	FCO	CDG	2016-05-02	13 :00	310

FIGURE 1 – Extrait de la table vol : vols de la compagnie Air France entre les aéroports Charles-de- Gaule (Paris) et Léonard-de-Vinci à Fiumicino (Rome)

— la table **aeroport** qui répertorie les aéroports européens ; elle contient les colonnes
  - `id_aero` : code de l'aéroport (chaine de caractères) ;
  - `ville` : principale ville desservie (chaine de caractères) ;
  - `pays` : pays dans lequel se situe l'aéroport (chaine de caractères).

Les types SQL date et time permettent de mémoriser respectivement un jour du calendrier grégorien et une heure du jour. Deux valeurs de type date ou de type time peuvent être comparées avec les opérateurs habituels (=, <, <=, etc.). La comparaison s'effectue suivant l'ordre chronologique. Ces valeurs peuvent également être comparées à une chaine de caractères correspondant à leur représentation externe ('aaaa-mm-jj' ou 'hh :mi').

id_aero	ville	pays
CDG	Paris	France
ORY	Paris	France
MRS	Marseille	France
FCO	Rome	Italie

FIGURE 2 – Extrait de la table aeroport

1. Écrire une requête SQL qui fournit le nombre de vols qui doivent décoller dans la journée du 2 mai 2016 avant midi.
2. Écrire une requête SQL qui fournit la liste des numéros de vols au départ d'un aéroport desservant Paris le 2 mai 2016.
3. Que fait la requête suivante ?

```sql
SELECT id_vol
FROM vol
  JOIN aeroport AS d ON d.id_aero = depart
  JOIN aeroport AS a ON a.id_aero = arrivee
WHERE
  d.pays = 'France' AND
  a.pays = 'France' AND
  jour = '2016-05-02'
```

4. Certains vols peuvent engendrer des conflits potentiels : c'est par exemple le cas lorsque deux avions suivent un même trajet, en sens inverse, le même jour et à un même niveau. Écrire une requête SQL qui fournit la liste des couples (Id1 , Id2) des identifiants des vols dans cette situation.

Allocation des niveaux de vol

Lors du dépôt d'un plan de vol, la compagnie aérienne doit préciser à quel niveau de vol elle souhaite faire évoluer son avion lors de la phase de croisière. Ce niveau de vol souhaité, le RFL pour requested flight level, correspond le plus souvent à l'altitude à laquelle la consommation de carburant sera minimale. Cette altitude dépend du type d'avion, de sa charge, de la distance à parcourir, des conditions météorologiques, etc.

Cependant, du fait des similitudes entre les différents avions qui équipent les compagnies aériennes, certains niveaux de vols sont très demandés ce qui engendre des conflits potentiels, deux avions risquant de se croiser à des altitudes proches. Les contrôleurs aériens de la région concernée par un conflit doivent alors gérer le croisement de ces deux avions.

Pour alléger le travail des contrôleurs et diminuer les risques, le système de régulation s'autorise à faire voler un avion à un niveau différent de son RFL. Cependant, cela engendre généralement une augmentation de la consommation de carburant. C'est pourquoi on limite le choix aux niveaux immédiatement supérieur et inférieur au RFL.

Ce problème de régulation est modélisé par un graphe dans lequel chaque vol est représenté par trois sommets. Le sommet 0 correspond à l'attribution du RFL, le sommet + au niveau supérieur et le sommet - au niveau inférieur. Chaque conflit potentiel entre deux vols sera représenté par une arête reliant les deux sommets concernés. Le coût d'un conflit potentiel (plus ou moins important en fonction de sa durée, de la distance minimale entre les avions, etc.) sera représenté par une valuation sur l'arête correspondante.

Dans l'exemple de la figure 3, faire voler les trois avions à leur RFL engendre un coût de régulation entre A et B de 100 et un coût de régulation entre B et C de 400, soit un coût total de la régulation de 500 (il n'y a pas de conflit entre A et C). Faire voler l'avion A à son RFL et les avions B et C au-dessus de leur RFL engendre un conflit potentiel de coût 100 entre A et B et 150 entre A et C, soit un coût total de 250 (il n'y a plus de conflit entre B et C).

On peut observer que cet exemple possède des solutions de coût nul, par exemple faire voler A et C à leur RFL et B au-dessous de son RFL. Mais en général le nombre d'avions en vol est tel que des conflits potentiels sont inévitables. Le but de la régulation est d'imposer des plans de vol qui réduisent le plus possible le coût total de la résolution des conflits.

Implantation du problème

Chaque vol étant représenté par trois sommets, le graphe des conflits associé à n vols v_0, v_1, …, v_{n-1} possède $3n$ sommets que nous numéroterons de 0 à $3n-1$. Nous conviendrons que pour $0 \leq k < n$:

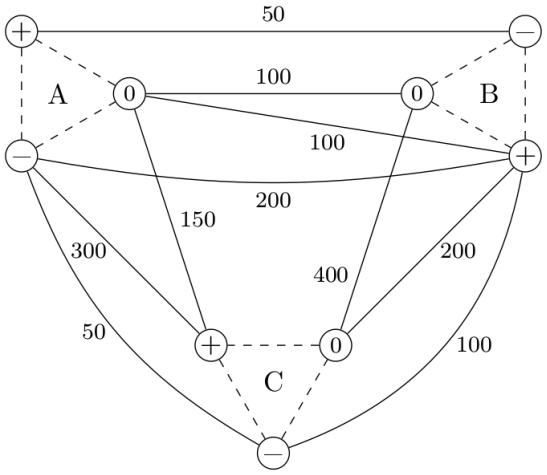

FIGURE 3 – Exemple de conflits potentiels entre trois vols

— le sommet $3k$ représente le vol v_k à son RFL ;
— le sommet $3k+1$ représente le vol v_k au-dessus de son RFL ;
— le sommet $3k+2$ représente le vol v_k au-dessous de son RFL ;

Le coût de chaque conflit potentiel est stocké dans une liste de $3n$ listes de $3n$ entiers (tableau $3n \times 3n$) accessible grâ ce à la variable globale `conflit` : si `i` et `j` désignent deux sommets du graphe, alors `conflit[i][j]` est égal au coût du conflit potentiel (s'il existe) entre les plans de vol représentés par les sommets `i` et `j`. S'il n'y a pas de conflit entre ces deux sommets, `conflit[i][j]` vaut `0`. On convient que `conflit[i][j]` vaut `0` si les sommets `i` et `j` correspondent au même vol (figure 4). On notera que pour tout couple de sommets (i, j), `conflit[i][j]` et `conflit[i][j]`, représentent un seul et même conflit et donc `conflit[i][j] == conflit[j][i]`.

```
conflit = [ [   0,   0,   0, 100, 100,   0,   0, 150,   0],
            [   0,   0,   0,   0,   0,  50,   0,   0,   0],
            [   0,   0,   0,   0, 200,   0,   0, 300,  50],
            [ 100,   0,   0,   0,   0,   0, 400,   0,   0],
            [ 100,   0, 200,   0,   0,   0, 200,   0, 100],
            [   0,  50,   0,   0,   0,   0,   0,   0,   0],
            [   0,   0,   0, 400, 200,   0,   0,   0,   0],
            [ 150,   0, 300,   0,   0,   0,   0,   0,   0],
            [   0,   0,  50,   0, 100,   0,   0,   0,   0] ]
```

FIGURE 4 – Tableau des coûts des conflits associé au graphe représenté figure 3

5. Écrire en Python une fonction `nb_conflits()` sans paramètre qui renvoie le nombre de conflits potentiels, c'est-à-dire le nombre d'arêtes de valuation non nulle du graphe.
6. Exprimer en fonction de n la complexité de cette fonction.

Régulation

Pour un vol v_k on appelle niveau relatif l'entier r_k valant 0, 1 ou 2 tel que :

— $r_k = 0$ représente le vol v_k à son RFL ;
— $r_k = 1$ représente le vol v_k au-dessus de son RFL ;
— $r_k = 2$ représente le vol v_k au-dessous son RFL.

On appelle régulation la liste $(r_0, r_1, ..., r_{n-1})$. Par exemple, la régulation $(0, 0, ..., 0)$ représente la situation dans laquelle chaque avion se voit attribuer son RFL. Une régulation sera implantée en Python par une liste d'entiers. Il pourra être utile d'observer que les sommets du graphe des conflits choisis par la régulation r portent les numéros $3k + r_k$ pour $0 \leq k < n$. On remarque également qu'au sommet s du graphe correspond le niveau relatif $r_k = s \, mod \, 3$ et le vol v_k tel que $k = \lfloor s/3 \rfloor$.

7. Écrire en Python une fonction `nb_vol_par_niveau_relatif(regulation)` qui prend en paramètre une régulation (liste de n entiers) et qui renvoie une liste de 3 entiers $[a, b, c]$ dans laquelle a est le nombre de vols à leurs niveaux RFL, b le nombre de vols au-dessus de leurs niveaux RFL et c le nombre de vols au-dessous de leurs niveaux RFL.

On appelle coût d'une régulation la somme des coûts des conflits potentiels que cette régulation engendre.

8. Écrire en Python une fonction `cout_regulation(regulation)` qui prend en paramètre une liste représentant une régulation et qui renvoie le coût de celle-ci.
9. Évaluer en fonction de n, la complexité de cette fonction.
10. Déduire de la question 8 une fonction `cout_RFL()` qui renvoie le coût de la régulation pour laquelle chaque vol est à son RFL.
11. Combien existe-t-il de régulations possibles pour n vols ? Est-il envisageable de calculer les coûts de toutes les régulations possibles pour trouver celle de coût minimal ?

L'algorithme Minimal

On définit le coût d'un sommet comme la somme des coûts des conflits potentiels dans lesquels ce sommet intervient. Par exemple, le coût du sommet correspondant au niveau RFL de l'avion A dans le graphe de la figure 3 est égal à $100 + 100 + 150 = 350$.

L'algorithme *Minimal* consiste à sélectionner le sommet du graphe de coût minimal ; une fois ce dernier trouvé, les deux autres niveaux possibles de ce vol sont supprimés du graphe et on recommence avec ce nouveau graphe jusqu'à avoir attribué un niveau à chaque vol.

Dans la pratique, plutôt que de supprimer effectivement des sommets du graphe, on utilise une liste `etat_sommet` de $3n$ entiers tels que :

— `etat_sommet[s]` vaut 0 lorsque le sommet s a été supprimé du graphe ;

— `etat_sommet[s]` vaut 1 lorsque le sommet s a été choisi dans la régulation ;

— `etat_sommet[s]` vaut 2 dans les autres cas.

12. Écrire en Python une fonction `cout_du_sommet(s, etat_sommet)` qui prend en paramètres un numéro de sommet s (n'ayant pas été supprimé) ainsi que la liste `etat_sommet` et qui renvoie le coût du sommet s dans le graphe défini par la variable globale `conflit` et le paramètre `etat_sommet`.
13. Exprimer en fonction de n la complexité de la fonction `cout_du_sommet`.
14. Écrire en Python une fonction `sommet_de_cout_min(etat_sommet)` qui, parmi les sommets qui n'ont pas encore été choisis ou supprimés, renvoie le numéro du sommet de coût minimal.
15. Exprimer en fonction de n la complexité de la fonction `sommet_de_cout_min`.
16. En déduire une fonction `minimal()` qui renvoie la régulation résultant de l'application de l'algorithme Minimal.
17. Quelle est sa complexité ? Commenter.

Opérations et fonctions Python disponibles

Fonctions

— `exp(x)` calcule l'exponentielle du nombre x

— `randint(n)` (n entier) renvoie un entier aléatoire compris entre 0 et n-1 inclus

— `random()` renvoie un nombre flottant tiré aléatoirement dans [0, 1[suivant une distribution uniforme

— `sqrt(x)` calcule la racine carrée du nombre x

— `time()` renvoie l'heure sous la forme d'un nombre de secondes depuis un instant de référence

Opérations sur les listes

— `u + v` construit une liste constituée de la concaténation des listes u et v : `[1, 2] + [3, 4, 5]` → `[1, 2, 3, 4, 5]`

— `n * u` construit une liste constitué de la liste u concaténée n fois avec elle-même : `3 * [1, 2]` → `[1, 2, 1, 2, 1, 2]`

— `u.append(e)` ajoute l'élément e à la fin de la liste u

— `del(u[i])` supprime de la liste u son élément d'indice i

— `u.insert(i, e)` insère l'élément e à la position d'indice i dans la liste u (en décalant les éléments suivants) ; si `i >= len(u)`, e est ajouté en fin de liste

— `len(u)` donne le nombre d'éléments de la liste u :

— `u[i], u[j] = u[j], u[i]` permute les éléments d'indice i et j dans la liste u